SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

[RELIABLE TRANSPORT LAYER PROTOCOL IN LOW PERFORMANCE 8-BIT MICROCONTROLLERS]

Background of Invention

[0001] BACKGROUND FIELD OF INVENTION

[0002] This invention relates to data transmission over a reliable transport layer protocol in a low processing power 8-bit microcontroller.

[0003] BACKGROUND DISCUSSION OF PRIOR ART

[0004] Communication between modern systems takes place through the use of a protocol layers model. Such model is usually called communication stack because each layer works on top of another. Each one has its own functions according to different levels of abstraction, so a given layer only has to be aware of its adjacent upper and lower neighbors. The stack begins with an Application layer (user related layer) at the top of the stack and finishes with a Link layer (related to the physical communication between systems) at the bottom before it reaches the physical medium.

[0005] One of the most common protocol stacks today is the TCP/IP (Transmission Control Protocol/Internet Protocol), which became popular due to the growth of the Internet. FIG. 1 depicts the typical four layers communication stack involved in an Internet connection.

[0006] Both source and destination systems (118, 128) have a four layer stack. It is usually called stack because a given layer always work on top of another layer,

beginning from the physical media up to the user interface. The bottom layer 110 is the link layer, which provides the network access and network sharing functions. On top of the link layer is the network layer 112, which handles the logical addressing of the existing systems on the network. The next layer is the transport layer 114. It handles the logical connections and the integrity of the information transfer. Finally, the application layer 116 represents the user related application working on the system. It dictates the length and content of the transmitted data.

[0007] The source system 118 must send information to another destination system 128, so the user application 126 passes the information to the TCP or UDP protocol 124.

[0008] The TCP protocol, located on the transport layer, is a connection-oriented protocol. Before any data transmission, the source node must ask for a connection to be opened at the destination node, that is to say, there must be a previous agreement between both systems previous to the data transfer.

[0009] After the connection is established, the information is divided in smaller blocks so the IP layer 122 could be able to handle it as an IP packet. To provide the integrity of the sent data, each block received at the destination node sends back an acknowledge packet to notify about the correct arrival of the information. A sequence number is provided by the source so the destination can tell which block of data is being acknowledged.

[0010] If the acknowledge packet doesn"t arrive, the source node makes a timeout and resends the unacknowledged packet. A retry counter defines the maximum number of resend packets.

[0011] In bi-directional communication, both source and destination systems must send and receive information and acknowledge packets. The TCP protocol supports piggybacking acknowledging, which is able to send back acknowledge packets carrying valid data information on it. With piggybacking acknowledging, unnecessary traffic is avoided since only one packet is sent instead of two.

[0012] The sequence number, besides providing the identification mechanism, also provides the order of the packets, so the TCP layer at the destination node is able to reconstruct the original sent information, in the correct order.

[0013] When all the information has been transferred, either the source or the destination node asks for a connection termination. Both systems agree to end the connection and the TCP transfer is successful.

[0014] There is another transport layer protocol called UDP (User Datagram Protocol). An UDP data transfer doesn"t establish a connection between the source and destination systems so every sent packet, called datagram, uses connectionless communication. This protocol doesn"t provide the acknowledge and ordering mechanism explained before in the TCP protocol thus every sent datagram is not guaranteed to arrive at the destination. Furthermore, it doesn"t provide the correct order of sent datagrams.

[0015] In contrast with the TCP protocol, UDP is not a reliable transport protocol, but it requires much less processing power and memory usage.

[0016] Every block sent from the TCP or UDP layer is received by the IP layer 122. This protocol is related to the addressing matters, placing the source and the destination addresses in the packet before it reaches the next layer. Those addresses are known as IP addresses. An IP address is a 32-bit number which identifies a node in the network.

[0017] The IP layer 122 is also able to fragment every block of information received from the previous layer, in case the network interface layer 120 is unable to handle the actual size of the packet. Generally such fragmentation can be disabled form the application layer. If the fragmentation is disabled and the packet is too big to be managed by the next layer (link layer), that packet is discarded.

[0018] Finally, the network interface layer 120 provides the platform to physically send the packet to the network.

[0019] At the destination node 128, its link layer 120b receives each piece of information and gives it to the IP layer 122b. If there has been any fragmentation during the transportation of the packets over the network, all the pieces are brought back together before it reaches the TCP (or UDP) layer 124b. In the case of receiving TCP blocks of information, the corresponding acknowledge packets are sent back to the source system. In the case of a UDP packet, the information gets to the final user application 126b with the same order it was received from the UDP layer 124b. In a

TCP connection, the information is correctly arranged before it reaches the application layer 126b.

[0020] With these four layers the information transfer can take place between two systems. The communication stack separates the type of systems involved in the communication from the communication process itself, being possible the existence of a high processing power system like a personal computer talking to a low processing power system like an 8-bit microcontroller. By those skilled in the art a low processing power 8-bit microcontroller can be considered working at 10 million cycles by second or 10MHz, with a maximum of 512 bytes in RAM (Random Access Memory).

[0021] In a personal computer the programming of the three upper layers (Application Layer 116, Transport Layer 114 and Network Layer 112) can be easily done, since that kind of systems have enough memory and processing power to handle a communication stack. The implementation of those same layers becomes a hard task if we consider the fact that 8-bit microcontrollers are very speed and memory limited to implement those three layers.

[0022] The resources needed for the stack (memory and processing time) are mostly used by the TCP protocol. A TCP connection consumes a lot of memory to maintain the connection related information (destination address, source and destination sequence number, etc) as well as a count of the acknowledged and unacknowledged sent blocks of data with their corresponding order in the sending-receiving mechanism. A complex algorithm is also needed to decode an incoming packet, since the connection state dictates the way to process that packet. For example, if the connection already has been established and a data packet is received, the TCP layer considers it as a valid data transfer and sends it to the application layer. If the connection has not been opened or the connection has been closed and the same data packet is received, it would be wasted away, since the actual connection state requires an opening transaction first.

[0023] For a personal computer, those resource requirements are easily met, but for low performance microcontrollers there is not enough memory and processing power to handle the TCP layer.

[0024] In the recent past there have been many attempts to provide a reliable transport protocol in the context of a low use of memory (e.g., US Patent 6161123). With this approach, a new reliable layer is added over the UDP layer and under the user application layer. Even when the amount of memory needed to provide such reliability is lower than a common TCP implementation, the new layer is still connection oriented. It means that both source and destination nodes must be always aware of the state of the connection and must be aware of which pieces of the information are being transferred. Certain memory consumption is still maintained and the decoding algorithm is still complex for an 8-bit microcontroller device.

[0025] Another attempt to provide a reliable transport mechanism (e.g., US Patent 6076114) also proposes an extra layer over the UDP protocol to ensure the integrity of the information. Again, the connection oriented extra layer and the existence of several connection states take us to the same problem of memory usage and complex decoding processing.

[0026] Both methods are suitable in the case of transmission of great amount of information, for example files transfer. However, the most common applications using low performance microcontrollers, in a networking context, only need short bursts of information. In a microcontroller-personal computer or a microcontroller-microcontroller connection scenario, the information traveling back and forward is very likely to contain small quantities of information, considering "low quantities"a number between 0 and 255 bytes. For example, it could contain commands to be executed by the microcontroller or messages indicating or asking the status of the device.

[0027] In summary, the memory and processing time required by a TCP protocol or any connection-oriented protocol are not suitable for a low processing power microcontroller.

Summary of Invention

[0028]

The present invention comprises a method which provides a reliable connectionless protocol to transfer short pieces of information. Such data transfer can take place between two microcontrollers or between a microcontroller and a personal

computer.

[0029] The proposed protocol works on top of the already existing UDP/IP protocols, which are intrinsically non-reliable. It provides a new communication layer of low memory and processing time usage, suitable for low processing power microcontroller.

[0030] OBJECTS AND ADVANTAGES

[0031] Accordingly, several objects and advantages of the present invention are:a)To provide a reliable method for information transfer.

[0032] b)To provide a simple and efficient method of information transfer suitable to the processing and memory limitations of a low processing power microcontroller.

[0033] c)To provide a connectionless method of information transfer, which minimizes the algorithm complexity and the needed processing time.

[0034] d)To provide an efficient method of information transfer suitable for those connections whose amounts of information transfer are in the range of 0 to 255 bytes, or message-oriented.

[0035] e)To provide a program, algorithm or mechanism such that the method could be programmed both on a microcontroller and a personal computer in a simple manner.

[0036] Other objects and advantages of this invention will become apparent from a consideration of the ensuing description and drawings.

Brief Description of Drawings

[0037] Fig 1 shows the communication layers involved in a typical TCP/IP-UDP/IP information transfer.

[0038] Fig 2 shows the typical length in bytes of an Ethernet/IP/(TCP-UDP) packet.

[0039] Fig 3 shows a UDP datagram including the RUDP two-fields header.

[0040] Fig 4 shows a piggybacking vs. a non-piggybacking information transfer.

[0041] Fig 5 shows the flowchart of the RUDP protocol send function.

[0042] Fig 6 shows the flowchart of the RUDP protocol receive function.

[0043] LIST OF REFERENCE NUMERALS IN DRAWINGS

[0044] 110 Communication Link layer

[0045] 112 Communication Network layer

[0046] 114 Communication Transport layer

[0047] 116 Communication Application layer

[0048] 118 Hypothetical source system A

[0049] 120 Network interface

[0050] 122 IP protocol

[0051] 124 TCP or UDP protocol

[0052] 126 User application

[0053] 128 Hypothetical destination system B

[0054] 210 Ethernet header

[0055] 212 IP protocol header

[0056] 214 UDP protocol header

[0057] 216 TCP protocol header

[0058] 218 TCP or UDP data field

[0059] 310 RUDP protocol header

[0060] 312 RUDP protocol type of packet field

[0061] 314 RUDP protocol Packet ID field

[0062] 316 RUDP data field

[0063] 414 Data sending process from A to B

[0064] 416 Data and acknowledge sending process from B to A

[0065] 418 Acknowledge sending process from A to B

[0066] 420 Data sending process from A to B

[0067] 422 Acknowledge sending process from B to A

[0068] 424 Data sending process from B to A

[0069] 426 Acknowledge sending process from A to B

[0070] 510 RUDP protocol

[0071] 512, 610 Start of function

[0072] 514, 520, 612, 616, 624, 626, 628, 634 Flowchart decision blocks

[0073] 516, 522, 614, 618, 630 Flowchart process blocks

[0074] 518, 524, 620, 622, 632 Flowchart end of function blocks

Detailed Description

[0075] Now, the present invention will be described by referring to the accompanying drawings that illustrate preferred embodiments of the invention.

[0076] A transfer process using the layers stack model shown on the Fig 1 has a nested scheme. The lower layers are the outer shells, so the information sent or received by the application layer becomes the content of the inner shell. Each shell added by a layer contains information handled only by that layer. That information is generally called layer header, or protocol header. Fig 2 shows the typical length, measured in bytes, of a UDP/IP and a TCP/IP packet sent over a LAN (Local Area Network) working with the Ethernet standard as the link layer protocol.

[0077] The Ethernet header 210 contains all the information needed to send a packet from a source to a destination system connected on the same LAN. The content of this header is irrelevant to this invention.

[0078] The IP header 212 contains the following fields: The total length of the IP packet, an identification number of the packet, one flag called "don"t fragment", indicating if the packet can (flag equals 0) or can"t be fragmented (flag equals 1), one flag called "more fragments"indicating the existence of more fragments (flag equals 1) from the same packet or indicating the presence of the last fragment (flag equals 0), the source IP address and the destination IP address. The other fields are not described due to its irrelevancy to the invention.

[0079] The UDP header 214 contains the source and destination ports and the length of the UDP packet. The UDP port provides a mechanism to maintain several logical connections to different applications working on the same system. The source system and the destination system can use different ports to communicate with each other, and that port number is specified in the UDP packet.

[0080] The TCP header 216 substitutes the UDP header 214 in the case of a connection-oriented transfer. The explanation of each field is not a matter of the invention. It was included to show the difference, in header length, between both transport protocols.The information being sent by the application is placed in the TCP or UDP data field 218.

[0081] The UDP transport protocol, as stated before, doesn"t provide a reliable transfer mechanism. By the inclusion of an intermediate transport layer called RUDP (Reliable-UDP), a reliable communication protocol is provided in a message-oriented information transfer.

[0082] Two new fields representing the RUDP protocol header must be added as part of the data field in a UDP datagram. Fig 3 shows a UDP datagram including the UDP header 214 and UDP data field 218. Two new fields have been included into the data field 218. Together, they form the RUDP header 310. The Type of packet field 312 is a one-byte value with three possible meanings:

[0083] · The packet contains valid data that must be acknowledged (reliable context). This value is called acknowledged service.

[0084] · The packet contains valid data that doesn"t need to be acknowledged (unreliable context). This value is called unacknowledged service · The packet is an acknowledge

response, so the data field is not valid. This value is called acknowledge service response.

[0085] The packet ID field 314 is a one-byte field. It has different meanings according to the Type of Packet field 312, as follows:

[0086] · If the packet contains valid data that must be acknowledged, the packet ID contains a number between 1 and 255 chosen by the source system. That number is sequentially increased with each send command executed by the application layer, whether the transfer is successful or not.

[0087] · If the packet contains valid data that doesn"t need to be acknowledged, this field is ignored.

[0088] · If the packet is an acknowledged response, this field contains the packet ID of the data packet that is being acknowledged.

[0089] The sent message is contained in the RUDP data field 316. The length of this field can be obtained subtracting two bytes (the length of the RUDP header) from the length field in the UDP header 214.

[0090] The conditions named before can be summarized in the following table: *Type of packet Packet IDData field* 0x01= Acknowledge serviceNumber between 1-255Valid data0x02=Unacknowledged serviceIgnoredValid data0x03=Acknowledge service response Previously received Packet ID between 1-255IgnoredThe RUDP layer works on top of the UDP layer and under the application layer. All information to be sent is received by the RUDP layer. The type of packet and packet ID fields are added as the beginning of the UDP data field. The type of packet must be the equivalent acknowledge service number (0x01) for reliable communication or the equivalent unacknowledged service number (0x02) for unreliable communication. The unacknowledged service has the same function as a normal UDP data transfer.

[0091] After sending a packet with the acknowledge service, a timer is turned on to wait for the arrival of the corresponding acknowledge service response (0x03) packet, whose packet ID field matches the original sent packet ID. If the timer expires, the packet is resent and the timer is reset and turned on again. This procedure is repeated

until an acknowledge arrives or until the maximum number of retries is reached. At this point the application layer is informed about the success or failure of the transfer. Any duplicated or out of time acknowledge response packet is discarded.

[0092] At the receiving system, an incoming acknowledge service packet always generates back an acknowledge service response packet. The new received packet is then checked in case it is a duplicated message. This is done by storing the packet ID and source address of the most recent received packets. If there is a match, it means the packet was already received before but the acknowledge response was lost; the incoming packet is ignored.

[0093] If the packet is valid (not duplicated), the message is passed to the application layer at the destination system.

[0094] An incoming unacknowledged service packet is not verified. It just goes up to the application layer at the destination system.

[0095] Communication in this invention considers two low processing power microcontrollers or a personal computer and a microcontroller as the source and destination systems. Most of the microcontroller related applications do not need great amounts if information. They are likely based on the transmission of short messages, considering a short message as a group of bytes in the range of 0 to 255 bytes, which dictates, for example, the mode of operation of the microcontroller. Applications involved with the handling of big pieces of information, like a file transfer mechanism, use more sophisticated equipment, capable of handling all that information in an efficient way.

[0096] In a short message context, some simplifications can be applied in this invention.

[0097] First, every packet is independent from the previous and subsequent packets, since all information is short enough to be contained in a single UDP/IP packet. In consequence, fragmentation of information at the IP layer is unnecessary. In the microcontroller, every sent packet must have the "don"t fragment"flag in the 1 state on the IP header in order to avoid fragmentation. It must also has the "more fragment" flag in the 0 state, indicating the existence of only one IP fragment. At the same time, every received packet must be checked for that same state in both flags. If any of the

flags are not in the indicated states, the packet is ignored.

[0098] Second, it is very unlikely to find a bi-directional communication. Most times, the transfer takes place in one direction. In consequence, there is no need of a piggybacking acknowledging mechanism and one packet ID field is enough for an exclusive non-piggybacking mechanism, instead of the two sequence numbers found on TCP for the received and transmitted data. In the case of a bi-directional communication (a sent message originates another returning message) there will be a penalty of one extra acknowledge message sent. Fig 4 shows both situations. In a piggybacking mechanism, the source system A 118 sends a packet to destination system B 128 (414). Destination B 128 sends back its message and the corresponding acknowledge to the source A 118 (416). The source A 118 sends the final acknowledge to destination B 128 (418). In this mechanism there is a total of three sent procedures.

[0099] In a non-piggybacking mechanism, the source A 118 sends a packet to destination B 128 (420). Destination B 128 sends back the acknowledge packet (422) followed by another packet containing his own message (424). The source A 118 receives both packets and sends back the corresponding acknowledge to destination B 128 (426). There is a total of four packets involved in this mechanism.

[0100] The penalty lies in an extra sent packet containing an acknowledgment and the delay time associated with the assembly of that packet. However, a forced non-piggybacking mechanism implies less complexity in both the sending and receiving mechanism, since de RUDP layer doesn"t need to be aware of any pending outgoing message to be sent with an acknowledge response, and neither it has to handle an incoming acknowledge response coming with a new message.

[0101] In third place, the protocol does not need to establish a connection between the source and the destination system.

[0102]

A connectionless scheme is enough in a message-oriented context, since the information messages are not related with each other. Thus the whole communication process is reduced to a sending mechanism and a receiving mechanism. A mechanism to open and close a connection (like the TCP protocol) is not needed anymore,

reducing the protocol"s algorithmic complexity.

[0103] Both functions can be clearly explained with a flowchart describing the algorithm needed to generate an outgoing message and the algorithm needed to decode an incoming message. Fig 5 shows the send function. The information or message to be sent comes from the user application 126 at the top of the stack to the RUDP layer 510. The provided algorithm 512-524 assemblies the packet and places it on the UDP layer 124. The corresponding header is added on the UDP 124, IP 122 and network interface 120 layers and finally the packet is sent over the network.

[0104] A complementary receive function shown on Fig 6 takes an incoming packet and decodes each header in an inverse order: Network interface 120 header first, followed by the IP 122 header and the UDP 124 header. The packet received by the RUDP layer 510 is finally decoded according to the algorithm 610-634.

[0105] OPERATION OF INVENTION

[0106] As said before, the invention takes in account two main processes, a sending function and a receiving function, being the receiving function the one with major complexity.

[0107] The sending function, as shown on figure 5, shows the user application 126, which commands the RUDP layer 510 to send a message. The process begins at 512. If the application layer asks for an unacknowledged service (514), the packet type takes the corresponding value of 0x03 (516) and the packet is sent to the UDP layer (518).

[0108] If the application layer asks for an acknowledge service (520), the packet type takes the corresponding value of 0x01, the retry timer is set and the packet ID takes the value in the ID_Counter (522). ID_Counter is an increasing counter which stores the actual value to be assigned to the packet ID. By sending sequential packet ID numbers, the destination system, if needed, can notice about the loss of a message. When it reaches the maximum value 255 it goes back to 1. Finally, the packet is sent to the next layer, the UDP layer (518).

[0109] If the user application asks for an unknown service number, an error warning is

sent back (524).

[0110] The UDP layer places the source port and the destination port. The source port is a 16-bit number indicating which port is available in the source system to receive any incoming packet. The destination port must be a 16 bit number known by the application layer. That port should be available at the destination system to receive the packet.

[0111] At the IP layer, the "don"t fragment"flag is set to 1, and the more fragment flag is set to 0. The source and destination addresses are placed and the packet is sent to the link layer where will be sent to the network.

[0112] The receiving function, shown on fig 6, begins with the acceptance of a packet from the network at the network interface layer 120. The original packet has headers from every layer. The link layer extracts the link layer header and passes the IP-UDP-RUDP-Application packet to the IP layer 122. This layer takes the IP header to check if the "don"t fragment"flag is set to the 1 state and the "more fragment" flag is set to the 0 state. If that is the case, the UDP-RUDP-Application packet is delivered to the next layer, UDP (124). Otherwise, the packet is discarded.

[0113] At the UDP layer 124 the destination port of the packet is extracted from the UDP header and is matched with the actual available open ports. If there is a match, the RUDP-Application packet is accepted for the next layer, the RUDP 510; otherwise, it"s discarded. Any subsequent response that should be sent to the system where the packet came from will go to the source port provided in the UDP header.

[0114] At the RUDP layer 510, the receive function starts (610) the decoding process. The type of packet field is first checked (612) for an acknowledge service message, identified by a 0x01. If that is the case, an acknowledge service response message is generated (614) by placing a 0x02 in the type of packet field and assigning the same packet ID as the received message. Finally, the response is sent to the UDP layer to be sent to the network.

[0115] The received message is then checked for duplicity (616). It is done by comparing the packet ID and the IP address of the source system with the packet ID"s-source IP addresses from the most recent received packets. If there is a match, the message is

ignored (622). If the message is not duplicated, the information about the ID packet and source IP address is stored (618) and the message is finally sent to the application layer (620).

[0116] If the type of packet turns out to be 0x02 (624), the message is an acknowledge service response to a previously sent message. The packet ID and the source IP address are matched to the values of the previously sent message (626, 628). If any of those values don"t match, the acknowledge response is wrong and it"s ignored (622).

[0117] When the acknowledge response is OK the retry timer is disabled and the ID_counter is incremented by one (630). The RUDP layer returns an OK value to the application layer 632 indicating the message transfer was successful.

[0118] Finally, when the type of packet is a 0x03 (634), the message is immediately delivered to the application layer (620), since the packet arrived as an unacknowledged service. Any RUDP packet with unknown packet type is ignored (622).

[0119] CONCLUSION, RAMIFICATIONS AND SCOPE OF INVENTION

[0120] Thus, the reader will see that the communication method of the invention provides a reliable, connectionless protocol, which minimizes the memory and processing time usage.

[0121] By working on top of a common UDP/IP communication stack, its implementation in a personal computer is simplified. Furthermore, by using a message-oriented instead of a connection-oriented scheme it is possible to implement the encoding-decoding algorithm in a low processing power 8-bit microcontroller with a minimum consumption of memory and processing time.

[0122] This invention shows a method for a reliable communication independent from the system on which is implemented. The present description particularly considers a personal computer and 8-bit microcontrollers as communication systems to show the flexibility of the implementation.

[0123] While our above description contains many specificities, these should not be construed as limitations to the scope of the invention, but rather as an exemplification

of one preferred embodiment thereof. Obviously, modifications and alterations will occur to others upon a reading and understanding of this specification such as, for example, several possible variations to the presented packet structure to include other fields into the protocol"s header, or changing the length or order of the specified fields.

[0124] The description above is intended, however, to include all such modifications and alterations insofar as they come within the scope of the appended claims or the equivalents thereof.